

PL/SQL

1.	Introducción	2
2.	Variables	2
3.	Control de flujo	3
3.1.	Instrucción condicional	4
3.2.	Bucles.....	6
4.	Cursores.....	8
5.	Excepciones	14
6.	Funciones.....	19
7.	Procedimientos.....	20
8.	Disparadores	21

1. Introducción

El lenguaje PL/SQL (Procedural Language Structured Query Language) es una extensión de programación a SQL que agrega ciertas construcciones propias de lenguajes estructurados obteniéndose como resultado un lenguaje más poderoso que SQL. Permite gestionar los datos de una base de datos ORACLE introduciendo instrucciones de control de flujo, definición de variables, utilización de procedimientos y funciones, manejo de excepciones,...

La unidad de programación utilizada por PL/SQL es el bloque. Cuando un bloque no tiene nombre se denomina anónimo. Otros tipos de bloques con nombre pueden ser procedimientos, funciones, disparadores (triggers),... En el caso de los bloques anónimos se trata básicamente de código fuente que reside en el lado del cliente y en el segundo caso es código que se almacena en la BD y está en el servidor.

Los bloques tienen la siguiente estructura

```
DECLARE
    Declaración de variables, cursores, excepciones, ...
BEGIN
    Instrucciones del bloque
[EXCEPTION
    Instrucciones para tratamiento de excepciones
]
END;
```

La sección DECLARE es opcional y contiene la declaración de las variables, cursores, excepciones,... del bloque. El ámbito de alcance de estos elementos es dentro del bloque en el que están declarados. La sección BEGIN contiene las instrucciones que se llevan a cabo dentro del bloque. Por último, la sección EXCEPTION es opcional y contiene las instrucciones para el tratamiento de las excepciones, generalmente correspondientes a errores en tiempo de ejecución.

2. Variables

Las variables en PL/SQL se declaran en la sección DECLARE y como argumentos de los procedimientos o funciones.

Los nombres de las variables deben empezar por una letra. El resto de caracteres pueden ser letras, números, \$, _ y #. La longitud máxima del nombre es de 30 caracteres. Es indiferente utilizar mayúsculas o minúsculas. Así por ejemplo, importe, Importe, e IMPORTE se refieren a la misma variable.

La sintaxis general para declarar una variable es

```
nombrevariable tipodatos;
nombrevariable tabla.columna%TYPE;
nombrevariable otravariable%TYPE;
```

Los tipos de datos que se pueden aplicar son los de SQL (NUMBER, CHAR, VARCHAR, DATE,...) y el tipo BOOLEAN, entre otros. Además, se pueden declarar variables asociándoles el mismo tipo de datos que el de una columna de una tabla de la BD o de otra variable utilizando %TYPE.

Ejemplos:

```
importe      NUMBER(6);
encontrado  BOOLEAN;
cantidad    ARTICULO.stock%TYPE;
disponible  encontrado%TYPE;
```

Para asignar valores a una variable se utiliza el operador “:=”. También se pueden asignar valores a una variable utilizando una consulta del tipo SELECT ... INTO ... que devuelva una única fila.

Ejemplos:

```
importe := 1000;
SELECT stock INTO cantidad FROM ARTICULO WHERE codigo = 'ART1';
```

Además de variables se pueden declarar constantes. La sintaxis para declarar y asignar el valor a una constante es

```
nombrevariable CONSTANT tipodatos := valor;
```

3. Control de flujo

En este apartado se especifican las instrucciones básicas de control de flujo referentes a las instrucciones condicionales y la creación de bucles o estructuras repetitivas. Este tipo de estructuras nos permiten modificar el flujo de ejecución de las instrucciones de un bloque dependiendo de determinadas condiciones.

Si no se introducen este tipo de instrucciones en la ejecución de un bloque se realizan todas las instrucciones del mismo de forma secuencial. En el Ejemplo 1 se presenta un bloque PL/SQL anónimo en el cual se realizan todas las instrucciones contenidas en el mismo de forma secuencial.

Ejemplo 1:

Imprimir la fecha de hoy, el stock de los artículos de códigos ART1 y ART2 y la suma de los stocks de ambos artículos. La estructura de la tabla ARTICULO es:

```
ARTICULO(codigo VARCHAR(6), stock NUMBER(3,0))
CP: CODIGO
```

```
SET SERVEROUTPUT ON
DECLARE
    hoy      DATE;
    stock1  ARTICULO.stock%TYPE;
```

```

stock2 ARTICULO.stock%TYPE;

BEGIN
    hoy := SYSDATE;
    SELECT stock INTO stock1 FROM ARTICULO WHERE codigo = 'ART1';
    SELECT stock INTO stock2 FROM ARTICULO WHERE codigo = 'ART2';

    DBMS_OUTPUT.PUT_LINE(TO_CHAR(hoy, 'DD-MM-YYYY'));
    DBMS_OUTPUT.PUT_LINE('ART1: ' || stock1);
    DBMS_OUTPUT.PUT_LINE('ART2: ' || stock2);
    DBMS_OUTPUT.PUT_LINE('Total articulos:' || (stock1+stock2));
END;
/

```

Veamos algunas instrucciones que se utilizan:

1. SERVEROUTPUT

Esta variable es una variable del sistema que indica si la salida se muestra por pantalla. Antes del procedimiento utilizamos la instrucción SET para establecer su valor a ON y así asegurarnos que los mensajes se imprimen por pantalla.

2. SYSDATE

Función que devuelve la fecha actual del ordenador.

3. DBMS_OUTPUT.PUT_LINE

La función PUT_LINE se utiliza para imprimir datos. Esta función se encuentra dentro del paquete DBMS_OUTPUT.

4. TO_CHAR

Función de conversión de tipos de datos. Esta función pasa a carácter el argumento que recibe. La función TO_CHAR tiene diferentes tipos de formatos de conversión. Otras funciones de conversión de datos son TO_NUMBER y TO_DATE.

5. Operador ||

Se utiliza para concatenar cadenas de caracteres. Al concatenar números generalmente se realiza la conversión directamente. En el caso de que la conversión dé algún fallo hay que convertir explícitamente el valor a carácter con TO_CHAR(valor). En el ejemplo anterior se podría haber hecho la conversión explícita con:

```
DBMS_OUTPUT.PUT_LINE('Total:' || TO_CHAR(stock1+stock2));
```

3.1. Instrucción condicional

Las instrucciones condicionales permiten ejecutar diferentes instrucciones dependiendo del valor de una condición. Dentro de las instrucciones condicionales están las estructuras IF y CASE.

La estructura general de la instrucción IF es:

```

IF condición THEN
    Instrucciones_siVerdadero
[ ELSIF condición2
    Instrucciones_siVerdadero_condición2]
[ ELSIF ... ]
[ ELSE
    Instrucciones ]
END IF;

```

Ejemplo 2:

Se quiere imprimir el stock actual del artículo ART1 y un mensaje “Bajo”, “Normal” o “Exceso”, dependiendo de cuál sea el valor de su stock actual. Por ejemplo, consideraremos que el stock es adecuado si está entre 5 y 10 unidades.

```

SET SERVEROUTPUT ON
DECLARE
    stock1 ARTICULO.stock%TYPE;

BEGIN
    SELECT stock INTO stock1 FROM ARTICULO WHERE codigo = 'ART1';

    DBMS_OUTPUT.PUT_LINE('Stock actual: ' || stock1);

    IF stock1 >= 5 AND stock1 <= 10 THEN
        DBMS_OUTPUT.PUT_LINE('Normal');
    ELSIF stock1 < 5 THEN
        DBMS_OUTPUT.PUT_LINE('Bajo');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Exceso');
    END IF;
END;
/

```

La instrucción CASE se utiliza para elegir entre diferentes posibilidades y ejecutar el conjunto de instrucciones que corresponden al primer valor que se evalúe como Verdadero.

La sintaxis general es:

```

CASE expresión
    WHEN valor1 THEN
        Instrucciones
    [ WHEN valor2 THEN
        Instrucciones ]
    [ ELSE
        Instrucciones ]
END CASE;

```

3.2. Bucles

Los bucles se utilizan para realizar repetidamente un conjunto de instrucciones hasta que se cumpla una determinada condición.

Para ejecutar un bucle podemos utilizar una de las siguientes estructuras:

LOOP	WHILE condición LOOP	FOR i IN [REVERSE] v1..v2
Instrucciones	Instrucciones	Instrucciones
EXIT WHEN condición;	END LOOP;	END LOOP;
END LOOP;		

Ejemplo 3:

Se quiere pedir un número por teclado e imprimir cuál es el valor de la suma de los números que van desde 1 hasta dicho número así como las sumas parciales. Por ejemplo, si el número que se introduce por teclado es 5 la salida es

```
1: 1
2: 3
3: 6
4: 10
5: 15
La suma final es: 15
```

En la primera línea se muestra la suma de 1 a 1. En la segunda línea se muestra la suma de 1 a 2, en la tercera línea la suma de 1 a 3: 6 y así hasta llegar al valor introducido por el usuario.

Para pedir un valor por teclado se utiliza el operador '&' junto con un nombre de identificador no definido. Este tipo de variables se llaman variables de sustitución y el sistema automáticamente pide el valor por pantalla.

Ejemplo 3: Solución con LOOP

```
SET SERVEROUTPUT ON
DECLARE
    i      NUMBER := 0;
    suma  NUMBER := 0;
    v      NUMBER;

BEGIN
    v := &numero;
    DBMS_OUTPUT.PUT_LINE('El valor introducido es: ' || v);
    LOOP
        EXIT WHEN i >= v;
        i := i + 1;
        suma := suma + i;
        DBMS_OUTPUT.PUT_LINE(i || ':' || suma);
    END LOOP;
```

```
DBMS_OUTPUT.PUT_LINE('La suma final es: ' || suma);
END;
```

Ejemplo 3: Solución con WHILE

```
SET SERVEROUTPUT ON
DECLARE
    i      NUMBER := 0;
    suma  NUMBER := 0;
    v      NUMBER;

BEGIN
    v := &numero;
    DBMS_OUTPUT.PUT_LINE('El valor introducido es: ' || v);
    WHILE i < v LOOP
        i := i +1;
        suma := suma + i;
        DBMS_OUTPUT.PUT_LINE(i || ':' || suma);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('La suma final es: ' || suma);
END;
/
```

Ejemplo 3: Solución con FOR

```
SET SERVEROUTPUT ON
DECLARE
    suma NUMBER := 0;
    v      NUMBER;

BEGIN
    v := &numero;
    DBMS_OUTPUT.PUT_LINE('El valor introducido es: ' || v);
    FOR i IN 1..v LOOP
        suma := suma + i;
        DBMS_OUTPUT.PUT_LINE(i || ':' || suma);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('La suma final es:' || suma);
END;
/
```

La ejecución de los bucles FOR se puede realizar en sentido contrario utilizando REVERSE. Para este ejemplo el resultado de la suma final es el mismo pero se realizaría la suma en sentido contrario, es decir desde el valor introducido hasta 1.

4. Cursos

Los cursos se utilizan para almacenar el resultado de una consulta (SELECT) sobre una BD que generalmente devuelve más de una fila. A partir de los resultados de la consulta se va recorriendo el cursor para acceder a cada uno de los registros/filas devueltas por la consulta.

Las acciones que se pueden realizar con los cursos son

1. Declarar.

```
CURSOR nombrecursor IS instrucción_SELECT;
CURSOR nombrecursor(parámetro1 tipodatos, parámetro2 tipodatos,...) IS
instrucción_SELECT;
```

Los cursos se declaran en la sección de declaración de variables de un bloque especificando la instrucción SELECT que tiene asociada.

Ejemplos:

```
CURSOR stockbajo IS
    SELECT codigo, stock
    FROM ARTICULO
    WHERE stock < 5;
```

```
CURSOR stocknormal IS
    SELECT codigo, stock
    FROM ARTICULO
    WHERE stock >= 5 AND stock <= 10;
```

2. Abrir.

```
OPEN nombrecursor;
```

Al abrir un cursor se lleva a cabo la consulta SELECT asociada al cursor y se almacena en memoria el resultado, situando el “puntero” de recorrido del cursor en la primera fila del conjunto de filas resultantes de la consulta.

3. Recorrer.

```
FETCH nombrecursor INTO variable1, variable2, ... ;
FETCH nombrecursor INTO variablefila;
```

Cada vez que se utiliza FETCH se almacenan los datos de la fila a la que apunta el puntero del cursor en las variables indicadas en INTO y se avanza el puntero a la siguiente fila. La instrucción FETCH se suele utilizar dentro de un bucle para recorrer todas las filas devueltas por la consulta y realizar las operaciones correspondientes.

Las variables donde se recogen los valores de una fila se pueden definir con un tipo de datos específico, con el tipo de datos de la columna correspondiente a la instrucción SELECT o como una única variable del tipo ROWTYPE. En este último caso la sintaxis para declarar la variable es:

```
nombrevariable nombrecursor%ROWTYPE;
```

y para acceder a cada una de las componentes de la fila se utiliza

```
nombrevariable.nombrecocomponente
```

4. Cerrar.

```
CLOSE nombrecursor;
```

Cuando ya no se utiliza el cursor lo cerramos y se libera la memoria utilizada.

Además de las funciones para el tratamiento de cursos, los cursos tienen una serie de atributos para controlar su estado, cuyos valores y significados son:

%ISOPEN TRUE si el cursor está abierto
 FALSE en caso contrario.

%NOTFOUND TRUE si el último FETCH no recupera ninguna fila.
 FALSE si el último FETCH devuelve una fila
 NULL si el cursor está abierto y no se ha realizado ningún FETCH.
 INVALID_CURSOR si el cursor no está abierto.

%FOUND TRUE si el último FETCH recupera una fila.
 FALSE si el último FETCH no devuelve una fila
 NULL si el cursor está abierto y no se ha realizado ningún FETCH.
 INVALID_CURSOR si el cursor no está abierto.

%ROWCOUNT Número total de filas recorridas por el cursor.
 INVALID_CURSOR si el cursor no está abierto.

Para utilizar estos atributos se utiliza el nombre del cursor delante del atributo, por ejemplo para saber si el cursor stockbajo está abierto se utiliza stockbajo%ISOPEN.

Ejemplo 4:

Realiza un listado de los artículos de los que se tenga un stock inferior a 8 unidades e imprima el código del artículo, stock actual y un mensaje “Bajo”, “Normal” o “Exceso”, dependiendo del stock. El listado estará ordenado ascendente por el stock del artículo. Al final del listado se imprimirá el número total de registros.

Un artículo tiene un stock “Bajo” si su stock es menor que 5, “Normal” si está entre 5 y 10 y en “Exceso” cuando es mayor que 10.

Ejemplo 4: Solución versión 1

En esta solución se utilizan dos variables denominadas codigofila y stockfila donde se guardan los valores del código y stock de cada artículo cada vez que se realiza la instrucción FETCH. Estas variables se definen del mismo tipo de datos que las correspondientes columnas en la tabla ARTICULO. Para recorrer el cursor se utiliza un bucle LOOP ... EXIT WHEN.

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR datos IS
        SELECT codigo, stock
        FROM ARTICULO
        WHERE stock < 8
        ORDER BY stock;

    codigofila ARTICULO.codigo%TYPE;
    stockfila ARTICULO.stock%TYPE;
    mensaje CHAR(10);

BEGIN
    OPEN datos;
    LOOP
        FETCH datos INTO codigofila, stockfila;
        EXIT WHEN datos%NOTFOUND;

        IF stockfila >= 5 AND stockfila <= 10 THEN
            mensaje := 'Normal';
        ELSIF stockfila < 5 THEN
            mensaje := 'Bajo';
        ELSE
            mensaje := 'Exceso';
        END IF;

        DBMS_OUTPUT.PUT_LINE(codigofila||' '||stockfila||' '||mensaje);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Total registros: '||datos%ROWCOUNT);
    CLOSE datos;
END;
/
```

Ejemplo 4: Solución versión 2

En este caso se define una única variable llamada fila del tipo ROWTYPE para guardar los datos que recoge la instrucción FETCH.

```

SET SERVEROUTPUT ON
DECLARE
    CURSOR datos IS
        SELECT codigo, stock
        FROM ARTICULO
        WHERE stock < 8
        ORDER BY stock;

    fila datos%ROWTYPE;
    mensaje CHAR(10);

BEGIN
    OPEN datos;
    LOOP
        FETCH datos INTO fila;
        EXIT WHEN datos%NOTFOUND;

        IF fila.stock >= 5 AND fila.stock <= 10 THEN
            mensaje := 'Normal';
        ELSIF fila.stock < 5 THEN
            mensaje := 'Bajo';
        ELSE
            mensaje := 'Exceso';
        END IF;

        DBMS_OUTPUT.PUT_LINE(fila.codigo||' '||fila.stock||' '||mensaje);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Total registros: ' || datos%ROWCOUNT);
    CLOSE datos;
END;
/

```

En lugar de utilizar las instrucciones OPEN, FETCH y CLOSE junto con un bucle para recorrer un cursor, se puede recorrer los datos de un cursor utilizando exclusivamente un bucle FOR del tipo

```

FOR nombrefila IN nombrecursor LOOP
    Instrucciones
END LOOP;

```

Este bucle abre el cursor nombrecursor, recupera las filas en la variable nombrefila y cierra el cursor.

Ejemplo 4: Solución versión 3

En este ejemplo se muestra cómo recorrer el cursor utilizando un bucle FOR. Ahora no hay que declarar la variable que recoge los registros del cursor sino que esta variable (en el ejemplo se llama fila) se especifica directamente en el bucle.

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR datos IS
        SELECT codigo, stock
        FROM ARTICULO
        WHERE stock < 8
        ORDER BY stock;

    mensaje CHAR(10);

BEGIN
    FOR fila IN datos LOOP
        IF fila.stock >= 5 AND fila.stock <= 10 THEN
            mensaje := 'Normal';
        ELSIF fila.stock < 5 THEN
            mensaje := 'Bajo';
        ELSE
            mensaje := 'Exceso';
        END IF;
        DBMS_OUTPUT.PUT_LINE(fila.codigo||' '||fila.stock||' '||mensaje);
    END LOOP;
END;
/
```

Al salir del bucle, no se puede acceder al atributo %ROWCOUNT del cursor porque el cursor está cerrado. Se podría definir una variable auxiliar para guardar el número de registros.

Además de definir cursosres en los cuales se utiliza una consulta SELECT con unos criterios fijos. Se pueden utilizar cursosres con parámetros de forma que cuando se ejecute la consulta se tomará el valor que tenga en ese momento el parámetro para ejecutar la consulta. Para definir un cursor con parámetros se utiliza la siguiente sintaxis

```
CURSOR nombrecursor(parámetro1 tipodatos, parámetro2 tipodatos,...) IS
    instrucción_SELECT;
```

Cuando se abra el cursor se cogerán los valores de los parámetros y se ejecuta la consulta asociada.

Ejemplo 5:

Modifica el ejemplo 4 para que ahora se seleccionen los artículos cuyo stock sea un valor que se introducirá utilizando un cursor con parámetros.

Veamos cómo quedarían las versiones 1 y 3 del Ejemplo 4, aplicando un cursor con parámetros y utilizando, por ejemplo, como valor mínimo el número 6. El texto “...” indica que se introducen las mismas instrucciones que en el correspondiente Ejemplo 4.

Ejemplo 5: Solución (Ejemplo 4 versión 1)

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR datos (minimo NUMBER) IS
        SELECT codigo, stock
        FROM ARTICULO
        WHERE stock < minimo
        ORDER BY stock;

    valor NUMBER;
    ...

BEGIN
    valor := 6;
    OPEN datos(valor);
    ...
END;
/
```

Ejemplo 5: Solución (Ejemplo 4 versión 3)

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR datos (minimo NUMBER) IS
        SELECT codigo, stock
        FROM ARTICULO
        WHERE stock < minimo
        ORDER BY stock;

    valor NUMBER;
    ...

BEGIN
    valor := 6;
    FOR fila IN datos(valor) LOOP
        ...
END;
/
```

5. Excepciones

Para manejar los posibles errores que se producen al ejecutar un bloque podemos dejar que los gestione el SGBD o bien controlarlos explícitamente utilizando la sección EXCEPTION de los bloques. La sintaxis de esta sección es:

```

EXCEPTION
    WHEN excepción1 THEN
        Instrucciones
    WHEN excepción2 THEN
        Instrucciones
    ...
    [WHEN OTHERS THEN
        Instrucciones a realizar en otro caso
    ]
END;
/

```

Se pueden tener excepciones internas de ORACLE (con nombre y sin nombre) y excepciones definidas por el usuario. Las excepciones internas con nombre tienen asignado un nombre y un número de error. Algunas de estas excepciones son:

Nombre	Número	Situación
CURSOR_ALREADY_OPEN	-06511	Se intenta abrir un cursor que ya está abierto.
DUP_VAL_ON_INDEX	-00001	Se intenta almacenar un valor duplicado en una columna que tiene un índice único.
INVALID_CURSOR	-01001	Se realiza una operación no válida sobre un cursor.
INVALID_NUMBER	-01722	Falla la conversión de una cadena de caracteres a un número.
LOGIN_DENIED	-01017	Se intenta conectar a ORACLE con un usuario o contraseña inválidos.
NO_DATA_FOUND	-01403	Si una instrucción SELECT que devuelve como mucho una fila, no devuelve ninguna.
PROGRAM_ERROR	-06501	Problema interno.
ROWTYPE_MISMATCH	-06504	Tipos incompatibles de datos en una asignación o un parámetro.
STORAGE_ERROR	-06500	No hay memoria suficiente o está corrupta.
TIMEOUT_ON_RESOURCE	-00051	Se excede el tiempo máximo de espera para un recurso.
TOO_MANY_ROWS	-01422	Si una instrucción SELECT que devuelve como mucho una fila, devuelve más de una.
VALUE_ERROR	-06502	Error aritmético, de conversión, truncamiento o redondeo, tamaño,...
ZERO_DIVIDE	-01476	División por cero.

Ejemplo 6:

El siguiente bloque muestra el código y stock de los artículos que tienen en stock el número de unidades que el usuario introduce por teclado (se utiliza una variable de sustitución para pedir dicho valor). Supongamos que la tabla ARTICULO tiene los siguientes datos:

codigo	stock
ART1	4
ART2	7
ART3	7
ART4	12

```
SET SERVEROUTPUT ON
DECLARE
    fila ARTICULO%ROWTYPE;

BEGIN
    SELECT * INTO fila
    FROM ARTICULO
    WHERE STOCK = &unidades;

    DBMS_OUTPUT.PUT_LINE('El artículo es:');
    DBMS_OUTPUT.PUT_LINE(fila.codigo||' '||fila.stock);
END;
/
```

¿Qué ocurre si introducimos 4 unidades? ¿Y si introducimos 5? ¿Y si son 7?

Podemos capturar las excepciones que se produzcan dentro de un bloque para que ORACLE no muestre el mensaje que tiene definido por defecto al producirse la excepción y tratar directamente nosotros la excepción. Para ello nos hace falta identificar qué excepción es, añadirla en la sección EXCEPTION e indicar las acciones a realizar. Por ejemplo podemos mostrar los mensajes “No hay artículos con ese stock” y “Hay más de un artículo con ese stock” cuando introduzcan un número que lleve a estas situaciones.

```
SET SERVEROUTPUT ON
DECLARE
    fila ARTICULO%ROWTYPE;

BEGIN
    SELECT * INTO fila
    FROM ARTICULO
    WHERE STOCK = &unidades;

    DBMS_OUTPUT.PUT_LINE('El artículo es:');
    DBMS_OUTPUT.PUT_LINE(fila.codigo||' '||fila.stock);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No hay artículos con ese stock');

    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Hay más de un artículo con ese stock');
```

```
END;
/
```

Nótese que en este ejemplo la consulta puede devolver más de una fila con lo cual en realidad sería más adecuado utilizar un cursor para que muestre todos los artículos que verifican el criterio de la consulta y no se produzca la excepción TOO_MANY_ROWS.

Ejercicio: Modifica el ejemplo para utilizar un cursor que tenga como parámetro el número de unidades y comprueba su funcionamiento en los 3 casos.

En el caso de las excepciones internas sin nombre se pueden capturar a partir de su número asociado. Los pasos a seguir son:

1. Declarar una variable del tipo EXCEPTION.

```
nombreexcepción EXCEPTION;
```

2. Asociar esta variable al número de la excepción.

```
PRAGMA EXCEPTION_INIT(nombreexcepción, númeroexcepción);
```

3. Incluir la excepción en la sección EXCEPTION.

Ejemplo 7:

El siguiente bloque se encarga de actualizar el stock del artículo 'ART1' sumándole el número de unidades que se introduce por teclado. Cuando la cantidad introducida hace que el stock sea negativo se ejecuta la excepción stock_negativo que emite un mensaje. Para que salte la excepción es necesario que la columna stock de la tabla ARTICULO tenga definida una restricción de que el stock debe ser mayor o igual que 0. En caso contrario, la excepción no saltará.

```
SET SERVEROUTPUT ON
DECLARE
    stock_negativo EXCEPTION;
    PRAGMA EXCEPTION_INIT(stock_negativo,-02290);
    stock1 ARTICULO.stock%TYPE;

BEGIN
    UPDATE ARTICULO
    SET stock = stock + &unidades
    WHERE codigo = 'ART1';

    COMMIT;

    SELECT stock INTO stock1
    FROM ARTICULO
    WHERE codigo = 'ART1';

    DBMS_OUTPUT.PUT_LINE('El stock actualizado es: ' || stock1);

```

```

EXCEPTION
  WHEN stock_negativo THEN
    DBMS_OUTPUT.PUT_LINE('No se realiza la actualización.');
    DBMS_OUTPUT.PUT_LINE('El stock final no puede ser menor que 0.');
END;
/

```

Cuando se detecta una excepción sin nombre podemos utilizar las funciones SQLCODE y SQLERRM para obtener el número y el mensaje de error asociado a dicho número. Suele ser útil capturarlas en la sección EXCEPTION con un apartado

```

WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Error (' || SQLCODE || '): ' || SQLERRM);

```

Además, de las excepciones internas, el programador puede definir sus propias excepciones. Para llevarlo a cabo podemos hacerlo de dos formas:

Opción 1:

1. Declarar una variable del tipo EXCEPTION.

```
nombreexcepción EXCEPTION;
```

2. Hacer “saltar” la excepción en el sitio correspondiente.

```
RAISE nombreexcepción;
```

3. Incluirla en el apartado EXCEPTION.

Opción 2:

Utilizar la función:

```
RAISE_APPLICATION_ERROR(númeroexcepción, mensaje);
```

El número que se asigna debe estar en el intervalo -20000 y -20999.

Es preferible utilizar la opción 1.

Ejemplo 8:

En el Ejemplo 7 se quiere añadir una excepción que se ejecute cuando no exista el artículo e imprima el mensaje “El artículo no existe.”.

Una posible solución según la opción 1 es definir la excepción código_noexiste y por ejemplo probarlo con un artículo inexistente “ARTIC1”. Para controlar cuándo falla la instrucción UPDATE utilizamos SQL%NOTFOUND, que hace referencia al atributo %NOTFOUND del último comando SQL introducido.

```

SET SERVEROUTPUT ON
DECLARE
  stock_negativo EXCEPTION;
  PRAGMA EXCEPTION_INIT(stock_negativo,-02290);
  stock1 ARTICULO.stock%TYPE;

```

```

articulo_noexiste EXCEPTION;

BEGIN

    UPDATE ARTICULO
    SET stock = stock + &unidades
    WHERE codigo = 'ARTIC1';

    IF SQL%NOTFOUND THEN
        RAISE articulo_noexiste;
    END IF;

    COMMIT;

    SELECT stock INTO stock1
    FROM ARTICULO
    WHERE codigo = 'ART1';

    DBMS_OUTPUT.PUT_LINE('El stock actualizado es: ' || stock1);

EXCEPTION
    WHEN stock_negativo THEN
        DBMS_OUTPUT.PUT_LINE('No se realiza la actualización.');
        DBMS_OUTPUT.PUT_LINE('El stock final no puede ser menor que 0.');
    WHEN articulo_noexiste THEN
        DBMS_OUTPUT.PUT_LINE('El artículo no existe.');
END;
/

```

Si queremos definir la excepción según la opción 2, una posible forma es

```

SET SERVEROUTPUT ON
DECLARE
    stock_negativo EXCEPTION;
    PRAGMA EXCEPTION_INIT(stock_negativo,-02290);
    stock1 ARTICULO.stock%TYPE;

BEGIN
    UPDATE ARTICULO
    SET stock = stock + &unidades
    WHERE codigo = 'ARTIC1';

    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20100,'El articulo no existe.');
    END IF;

    COMMIT;

EXCEPTION
    WHEN stock_negativo THEN
        DBMS_OUTPUT.PUT_LINE('No se realiza la actualización.');
        DBMS_OUTPUT.PUT_LINE('El stock final no puede ser menor que 0.');
END;
/

```

6. Funciones

Las funciones tienen como finalidad realizar un cálculo y devolver un valor y se almacenan en la BD.

La sintaxis para definir una función es:

```
CREATE OR REPLACE FUNCTION nombrefunción (parámetros)
RETURN tipodatos
IS | AS
    Declaramos las variables
BEGIN
    Instrucciones del bloque
[EXCEPTION
    Instrucciones para tratamiento de excepciones
]
END;
```

Si se produce algún error se puede consultar la vista USER_ERRORS para ver cuál es el error cometido indicando el nombre de la función en mayúsculas y entre comillas simples.

```
SELECT * FROM USER_ERRORS WHERE NAME='NOMBREFUNCION' ;
```

Una vez creada la función sin errores se puede llamar desde cualquier bloque o directamente en la línea de comandos con

```
SELECT nombrefunción FROM DUAL;
```

Ejemplo 9:

La función TOTAL_ARTICULOS obtiene el número total de artículos en stock que hay en la tabla ARTICULO.

```
CREATE OR REPLACE FUNCTION TOTAL_ARTICULOS
RETURN NUMBER
IS
    total NUMBER := 0;

BEGIN
    SELECT SUM(stock) INTO total
    FROM ARTICULO;

    RETURN total;
END;
/
```

Para ejecutar esta función desde la línea de comandos utilizamos

```
SELECT TOTAL_ARTICULOS FROM DUAL;
```

Ejemplo 10:

La siguiente función recibe como parámetro el código de un artículo y devuelve el número de unidades en stock.

```
CREATE OR REPLACE FUNCTION STOCK_ARTICULO(codart ARTICULO.codigo%TYPE)
RETURN NUMBER
IS
    total NUMBER := 0;

BEGIN
    SELECT stock INTO total
    FROM ARTICULO
    WHERE codigo = codart;

    RETURN total;
END;
/
```

Si queremos ejecutar la función desde la línea de comandos para el artículo ART1, escribimos

```
SELECT STOCK_ARTICULO('ART1') FROM DUAL;
```

Para eliminar una función se utiliza la instrucción

```
DROP FUNCTION nomrefunción;
```

La vista USER_PROCEDURES contiene información sobre los procedimientos y funciones creados por un usuario. Para consultar las funciones y procedimientos que contiene se puede ejecutar una consulta SELECT sobre esta vista.

7. Procedimientos

Los procedimientos se utilizan para realizar una serie de acciones sobre la BD y se almacenan en la base de datos. Puede tener parámetros de entrada (IN), salida (OUT) y de entrada y salida (IN OUT). Si no se especifica nada el parámetro es de entrada.

Su sintaxis es:

```
CREATE OR REPLACE PROCEDURE nombreprocedimiento (parámetros)
IS | AS
    Declaramos las variables
BEGIN
    Instrucciones

[EXCEPTION
    Instrucciones para tratamiento de excepciones
]
END;
```

Para ejecutar un procedimiento se utiliza la instrucción

```
EXECUTE nombreprocedimiento
```

Para eliminar una función se utiliza la instrucción

```
DROP PROCEDURE nombreprocedimiento;
```

Ejemplo 11:

El siguiente procedimiento recibe como argumento el código del artículo y el número de unidades y actualiza el stock de dicho artículo en la cantidad de unidades que se le indique. Si al actualizar se obtiene un valor menor que 0 se imprime un mensaje.

```
CREATE OR REPLACE
PROCEDURE REPONER_STOCK (codart ARTICULO.codigo%TYPE, unidades IN NUMBER)
IS
    stock_negativo EXCEPTION;
    PRAGMA EXCEPTION_INIT(stock_negativo,-02290);

BEGIN
    UPDATE ARTICULO
    SET stock = stock + unidades
    WHERE codigo = codart;

    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20100,'El articulo no existe.');
    END IF;
    COMMIT;
EXCEPTION
    WHEN stock_negativo THEN
        DBMS_OUTPUT.PUT_LINE('El stock no puede ser menor que 0.');
END;
/
```

Para ejecutar el procedimiento para el artículo ‘ART1’ con 5 unidades escribimos

```
EXECUTE REPONER_STOCK('ART1',5);
```

8. Disparadores

Un disparador (*trigger*) es un código que se ejecuta automáticamente cuando se pretende realizar una instrucción sobre la BD. Se pueden diferenciar disparadores asociados a tablas, vistas y al sistema. En nuestro caso, trataremos únicamente los disparadores asociados a tablas o vistas. En estos casos, el disparador es código PL/SQL que se ejecuta al realizar una operación del tipo DML sobre una tabla o vista de la BD.

Entre las aplicaciones de los disparadores están:

- Crear reglas de integridad que son difíciles de definir mediante restricciones (*constraints*).
- Realizar cambios en la BD de forma transparente al usuario.

- Validar datos y evitar errores.

La sintaxis para crear un disparador es:

```

CREATE OR REPLACE TRIGGER nombretrigger
{BEFORE | AFTER | INSTEAD OF}
    { INSERT | DELETE | UPDATE [OF columnas]
      [OR { INSERT | DELETE | UPDATE [OF columnas] } ] }
ON nombrertabla
[FOR EACH ROW [WHEN (condición) ] ]

[DECLARE
    Declaramos las variables
]

BEGIN
    Instrucciones

[EXCEPTION
    Instrucciones para tratamiento de excepciones
]
END;

```

Para crear un disparador se distinguen tres partes:

1. Evento.

Se refiere al comando DML (INSERT, DELETE, UPDATE) que hace que se ejecute el disparador y a la tabla asociada. Hay que indicar cuándo se ejecuta el disparador con respecto al comando DML:

- BEFORE: antes de ejecutar el comando
- AFTER: después de ejecutar el comando
- INSTEAD OF: se sustituye el comando DML por las instrucciones que se indiquen en el disparador (para vistas que no permitan instrucciones DML).

2. Tipo de disparador.

Puede ser un disparador tipo fila (FOR EACH ROW) en el que para cada una de las filas (se puede especificar en la parte WHEN que las filas cumplan una condición) se ejecutan las instrucciones del disparador, o bien puede ser un disparador tipo instrucción que ejecuta las instrucciones del disparador una sola vez cuando se produce el evento que lanza el disparador.

3. Acción.

Esta parte se corresponde a las acciones que lleva a cabo el disparador. Se implementan en código PL/SQL dentro de la sección DECLARE ... END.

Para referirse al valor nuevo y al antiguo de una columna de un registro de una tabla se utiliza respectivamente :NEW.nombrerolumna y :OLD.nombrerolumna, donde nombrerolumna es el nombre de la columna correspondiente.

NOTA: Cuando se quiere hacer referencia a estos valores en la condición WHEN de FOR EACH ROW **no** se utiliza ':' delante de NEW y OLD.

En la siguiente tabla se muestra cuándo están disponibles las variables :NEW y :OLD de una columna para referenciar a sus valores nuevos y antiguos/actuales, respectivamente.

Evento	NEW	OLD
INSERT	Sí	NULL
DELETE	NULL	Sí
UPDATE	Sí	Sí

Por otro lado, como se puede ver en la sintaxis de la creación de un disparador, se pueden definir disparadores que se activen debido a varios eventos (utilizando OR). En estos casos, si se quiere diferenciar dentro de la sección BEGIN ... END qué evento ha lanzado el disparador, se utilizan los comandos INSERTING, DELETING, UPDATING y UPDATING(nombrecolumna) que devuelven un valor verdadero si el disparador se ha activado por una instrucción INSERT, DELETE, UPDATE o UPDATE con actualización de la columna nombrecolumna, respectivamente. Generalmente estos comandos se utilizan junto con una instrucción condicional (IF).

Algunas operaciones de mantenimiento sobre los disparadores son:

- Activar/Desactivar

Cuando se crea un disparador por defecto está activado (ENABLE). Si se quiere que el disparador no se ejecute hay que asignar su estado a desactivado (DISABLE).

Para activar/desactivar un disparador

```
ALTER TRIGGER nombretrigger {ENABLE | DISABLE}
```

Para activar/desactivar todos los disparadores de una tabla se utiliza

```
ALTER TABLE nombretabla {ENABLE | DISABLE} ALL TRIGGERS
```

- Eliminar

```
DROP TRIGGER nombretrigger
```

- Consultar

La información sobre los disparadores creados por un usuario se encuentra en la vista USER_TRIGGERS. Con el comando DESCRIBE se puede ver la estructura de la vista y realizar la consulta (SELECT) sobre los campos que interesen.

A continuación, se muestran algunos ejemplos de disparadores.

Ejemplo 12:

El siguiente disparador se ejecuta cada vez que se produce una inserción o actualización en la tabla ARTICULO y almacena en la tabla ARTICULO_MOVIMIENTO los datos del artículo, la fecha, el tipo de movimiento realizado (I: insertar, M:modificar) y el usuario que lo ha hecho.

Estructura de las tablas:

ARTICULO(codigo VARCHAR(6), stock NUMBER(3,0))
CP: codigo

ARTICULO_MOVIMIENTO(codigo VARCHAR(6), stock NUMBER(3,0), fecha DATE,
accion VARCHAR(1), usuario VARCHAR(15))

```
CREATE OR REPLACE TRIGGER MODIFICAR_ARTICULO
  BEFORE INSERT OR UPDATE ON ARTICULO
  FOR EACH ROW

DECLARE
  tipo_movimiento VARCHAR(1);

BEGIN
  IF INSERTING THEN
    tipo_movimiento := 'I';
  ELSE
    tipo := 'M';
  END IF;

  INSERT INTO ARTICULO_MOVIMIENTO VALUES
  (:NEW.codigo, :NEW.stock, SYSDATE, tipo_movimiento, USER);
END;
/
```

Ejemplo 13:

Se tienen tres tablas llamadas ARTICULO, PEDIDO y SUMINISTRAR que guardan información sobre los artículos, los pedidos que se realizan y los artículos pendientes de suministro. La estructura de las tablas es:

ARTICULO(codigo VARCHAR(6), stock NUMBER(3,0))
CP: codigo

PEDIDO(numero NUMBER, codart VARCHAR(6), unidades NUMBER(3,0), estado
VARCHAR(15))
CP: numero
Cajena: codart → ARTICULO

SUMINISTRAR(numpedido NUMBER, codart VARCHAR(6), unidades NUMBER(3,0),
usuario VARCHAR(15), fecha DATE)

Se quiere que cuando se dé de alta un pedido si hay suficientes unidades en stock se actualice el stock en la tabla ARTICULO, restando el número de unidades puestas. En caso de no tener suficientes unidades en stock, se cogen las unidades disponibles en stock, se da de alta un nuevo registro en la tabla SUMINISTRAR que hace referencia al pedido del artículo incluyendo el número de pedido al que se refiere, el código del artículo , el número de unidades que faltan para completar el total de unidades del pedido y el usuario y fecha en que se produce. En estos casos, en los cuales no se puede completar el pedido inicial se asigna el estado del pedido el valor “Pendiente stock númerounidades”.

```

CREATE OR REPLACE TRIGGER PEDIR_ARTICULO
  BEFORE INSERT ON PEDIDO
  FOR EACH ROW

DECLARE
  stockactual      NUMBER;
  stockfaltante   NUMBER;

BEGIN
  SELECT stock INTO stockactual
    FROM ARTICULO
   WHERE ARTICULO.codigo = :NEW.codart;

  IF stockactual >= :NEW.unidades THEN
    UPDATE ARTICULO
      SET stock = stockactual - :NEW.unidades
     WHERE codigo = :NEW.codart;
  ELSE
    UPDATE ARTICULO
      SET stock = 0
     WHERE codigo = :NEW.codart;

    stockfaltante := :NEW.unidades - stockactual;
    :NEW.estado := 'Pendiente stock ' || stockfaltante;

    INSERT INTO SUMINISTRAR VALUES
      (:NEW.numero, :NEW.codart, stockfaltante, USER, SYSDATE);
  END IF;
END;
/

```

Ejemplo 14:

Una empresa dispone de la información sobre los empleados y departamentos a los que pertenecen.

```

DEPARTAMENTO(codigo VARCHAR(10), nombre VARCHAR(20));
CP: codigo

EMPLEADO(dni VARCHAR(10) , nombre VARCHAR(20), apellido1 VARCHAR(20),
depto VARCHAR(10))
CP: dni
Cajena: depto → DEPARTAMENTO

```

Se quiere que solamente se pueda eliminar un departamento si tiene 1 o 2 empleados. En ese caso se borra el departamento y todos sus empleados. En caso contrario, no se realiza el borrado y se imprime el mensaje “El departamento código no se puede borrar. Total empleados: número”.

```
SET SERVEROUTPUT ON
CREATE OR REPLACE TRIGGER ELIMINAR_DEPTO
    BEFORE DELETE ON DEPARTAMENTO
    FOR EACH ROW

DECLARE
    totalemployees NUMBER;

BEGIN
    SELECT COUNT(*) INTO totalemployees
    FROM EMPLEADO
    WHERE EMPLEADO.depto = :OLD.codigo;

    IF totalemployees > 2 THEN
        RAISE_APPLICATION_ERROR(-20001,'El departamento ' || :OLD.codigo || 
        ' no se puede borrar. Total empleados: ' || totalemployees);
    ELSE
        DELETE FROM EMPLEADO
        WHERE EMPLEADO.depto = :OLD.codigo;
    END IF;
END;
/
```